CYBERTEC
DATA SCIENCE & POSTGRESQL

When it all GOes right

Writing applications with Go for PostgreSQL

When it all GOes right

Pavlo Golub

Senior Consultant/Developer

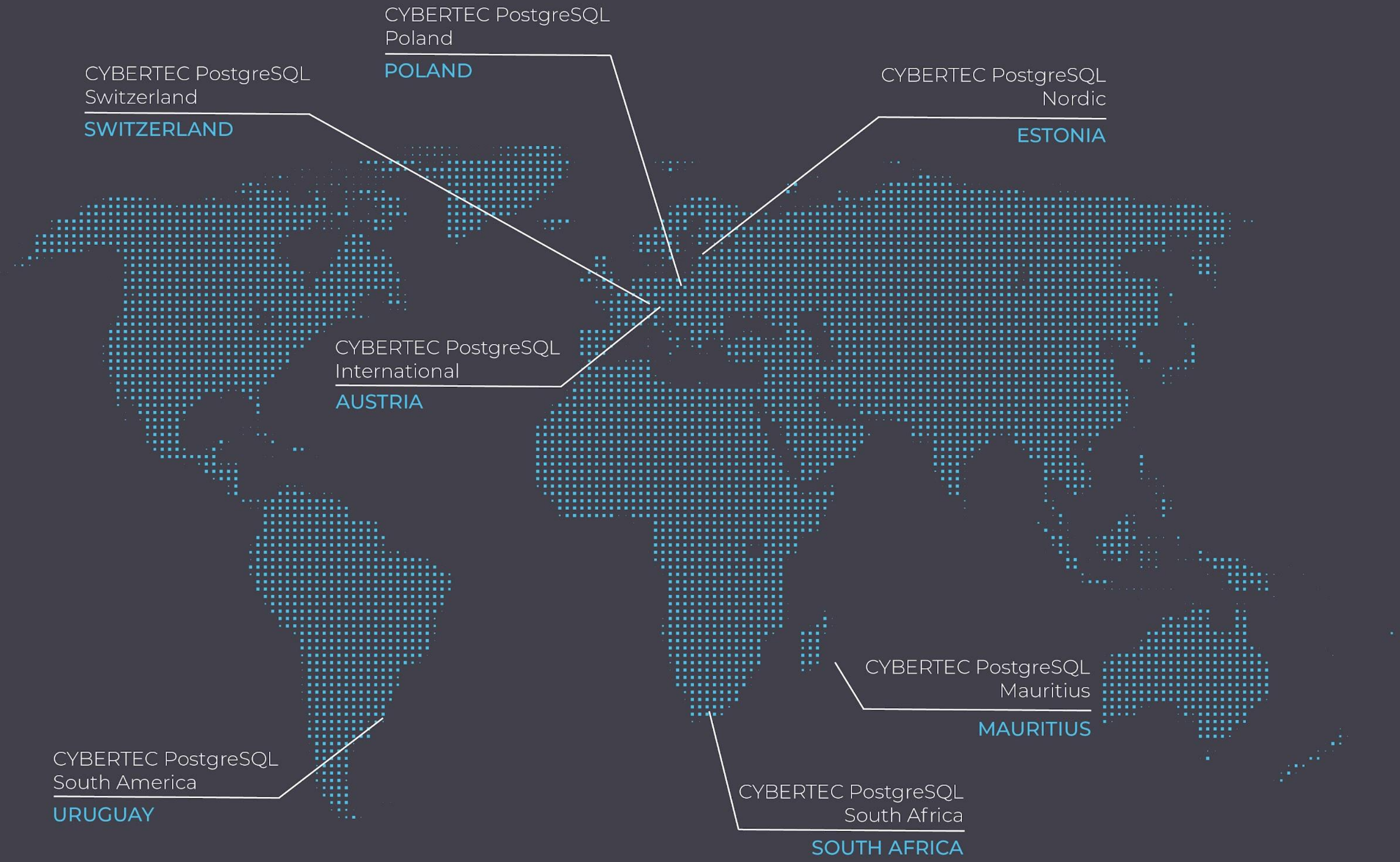
pavlo.golub@cybertec.at

@PavloGolub

CYBERTEC
DATA SCIENCE & POSTGRESQL

About CYBERTEC

- Inhouse development
- International team of developers
- Specialized in data services
- Owner-managed since 2000


CYBERTEC PostgreSQL Poland
POLAND
CYBERTEC PostgreSQL Switzerland
SWITZERLAND
CYBERTEC PostgreSQL Nordic
ESTONIA
CYBERTEC PostgreSQL International
AUSTRIA
CYBERTEC PostgreSQL Mauritius
MAURITIUS
CYBERTEC PostgreSQL South America
URUGUAY
CYBERTEC PostgreSQL South Africa
SOUTH AFRICA

CYBERTEC
DATA SCIENCE & POSTGRESQL

CLIENT SECTORS

- ICT
- University
- Government
- Automotive
- Industry
- Trade
- Finance
- etc.

Klarna.
BMW
hp
amazon
bon prix
Atos
skype™
Lufthansa
WILLHABEN
voestalpine
SPAR
SBB Cargo
hims
PORSCHE
IBM®
NOKIA
ÖBB
Audi
METRO
MAGNA
SIEMENS
Magenta
VOR
DER VERKEHRSVERBUND
ALCATEL

CYBERTEC
DATA SCIENCE & POSTGRESQL

DATABASE SERVICES

DATA SCIENCE

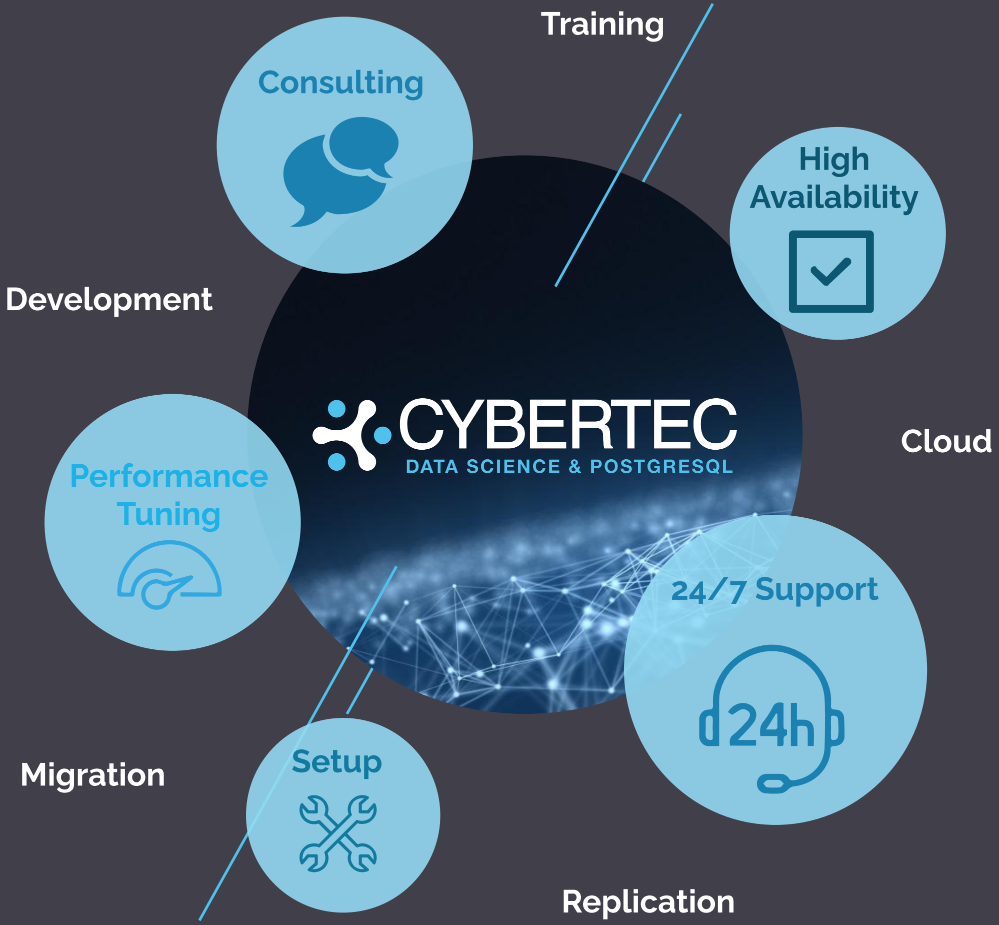
- Artificial Intelligence
- Machine Learning
- Big Data
- Business Intelligence
- Data Mining
- etc.

POSTGRESQL

- 24/7 Support
- Training
- Consulting
- Performance Tuning
- Clustering
- etc.

POSTGRESQL DATABASE SERVICES

- 24/7 Support
- Training
- Consulting
- Performance Tuning
- Clustering
- etc.



CYBERTEC
DATA SCIENCE & POSTGRESQL

DATA SCIENCE

- Artificial Intelligence
- Machine Learning
- Big Data
- Business Intelligence
- Data Mining
- etc.

CYBERTEC
DATA SCIENCE & POSTGRESQL

CYBERTEC
DATA SCIENCE & POSTGRESQL

ADVANTAGES of PostgreSQL

MOST ADVANCED OPEN SOURCE DATABASE SYSTEM

25 YEARS OF DEVELOPMENT

NO LICENSE COSTS

EXTENSIVE FUNCTIONALITY

LOW SUPPORT COSTS

RELIABILITY

SCALABILITY

Today’s agenda

- Intro to Go
- IDEs and tools
- Drivers
- Useful extensions
- Testing

CYBERTEC
The PostgreSQL Database Company

When it all GOes right

Intro to the Go

CYBERTEC
DATA SCIENCE & POSTGRESQL

- ✓ NATIVE BINARIES
- ✓ SIMPLICITY
- ✓ FAST COMPILATION
- ✓ STANDARD LIBRARY
- ✓ CONCURRENCY
- ✓ EASY TO LEARN
- ✓ COMPREHENSIVE TOOLS

WHY
Go

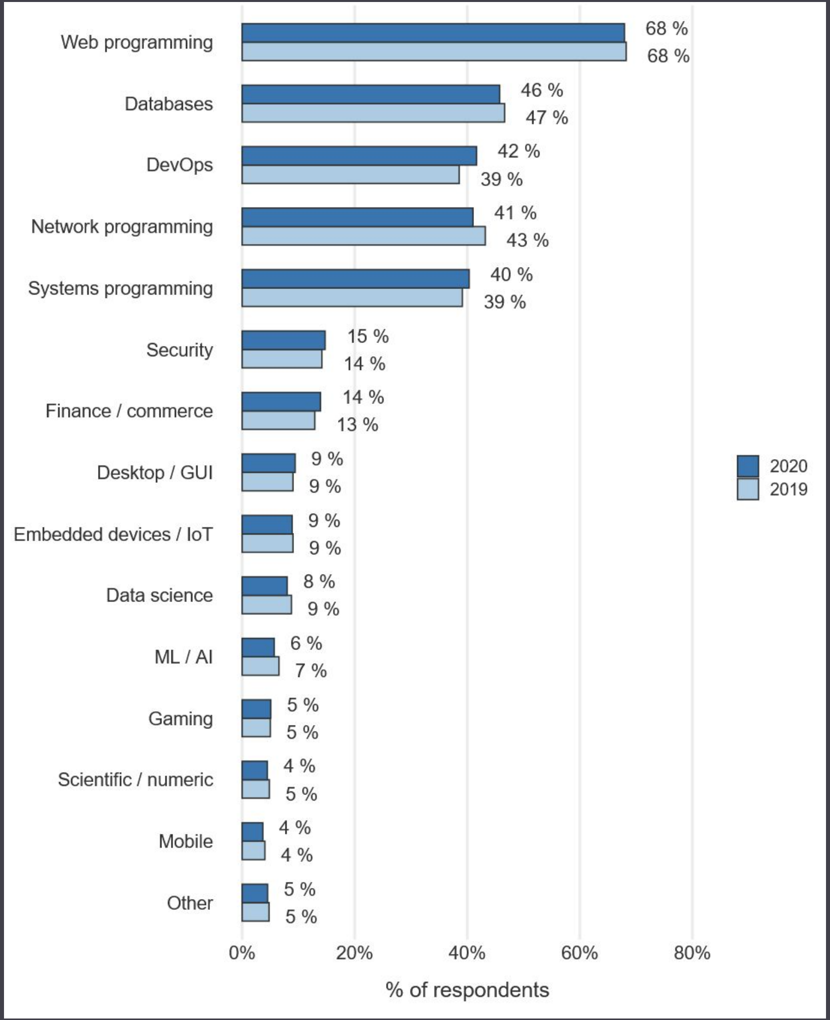
Go is an open source programming language that makes it easy to build simple, reliable, and efficient software. Ii was built to resemble a simplified version of the C programming language. It compiles at the machine level. Go was created at Google in 2007 by Robert Griesemer, Rob Pike, and Ken Thompson.

CYBERTEC
DATA SCIENCE & POSTGRESQL

I work with Go in the following areas:

“ We also asked about larger areas in which respondents work with Go. The most common area by far was web development (68%), but other common areas included databases (46%), DevOps (42%) network programming (41%) and systems programming (40%).

https://go.dev/blog/survey2020-results



Top products written in Go

- **Kubernetes (K8s)** - production-grade container management
- **Moby** - a collaborative project for the container ecosystem
- **Hugo** - the world’s fastest framework for building websites
- **Grafana** - observability and data visualization platform
- **Gogs** - painless self-hosted Git service
- **Etcd** - distributed reliable key-value store
- **Caddy** - fast, multi-platform web server with automatic HTTPS

Top products written in Go
PostgreSQL-related

- **pgweb** - cross-platform client for PostgreSQL databases
- **stolon** - a cloud native PostgreSQL manager
- **postgres operators** by Zalando and by Crunchy
- **wal-g** - Archival and Restoration for Postgres
- **pgcenter** - top-like admin tool for troubleshooting Postgres
- **pgwatch2** - PostgreSQL metrics monitor/dashboard
- **pg_timetable** - an advanced scheduling for PostgreSQL
- **pg_flame** - a flamegraph generator for EXPLAIN output

Top products written in Go

- Find more on **https://github.com/avelino/awesome-go**

When it all GOes right

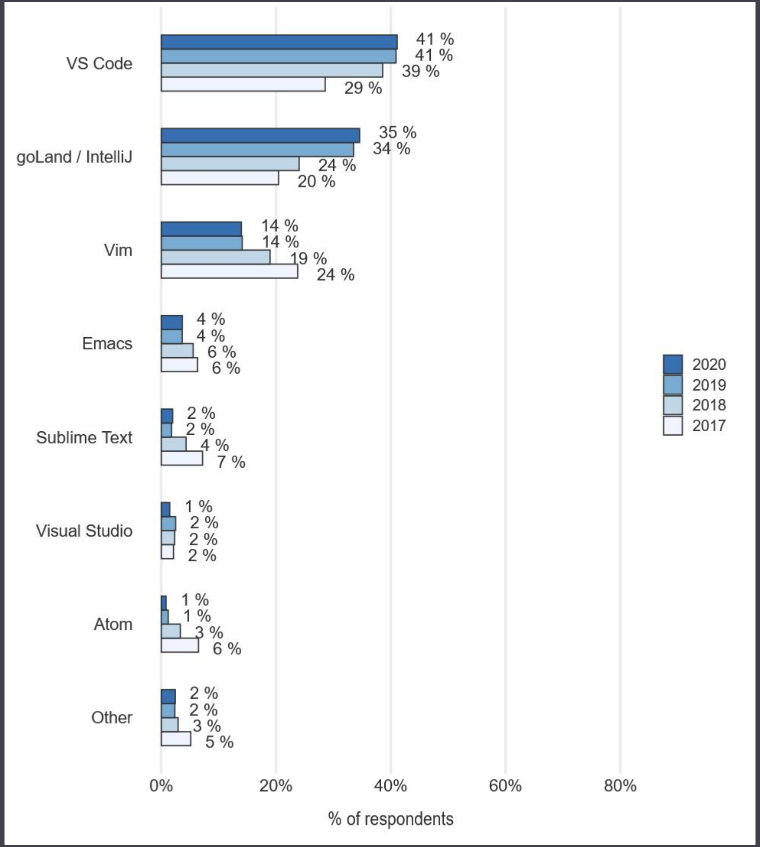
IDEs and tools

CYBERTEC
DATA SCIENCE & POSTGRESQL

The most popular editors

- **Visual Studio Code** by Microsoft
- **GoLand** by JetBrains
- **Vim & Neovim**
- **Emacs**
- **Sublime Text**
- other

https://go.dev/blog/survey2020-results



CYBERTEC
The PostgreSQL Database Company

My environment

- **VSCode** with the official vscode-go plugin
- **rakyll/gotest** - `go test` but with colors
- **golangci-lint** - fast linters Runner for Go
- **Tabnine** - the AI code completion tool
- **GoReleaser** - deliver Go binaries as fast and easily as possible
- **PostgreSQL** - most advanced object-relational database
- **gitpod.io** - container-based ready-to-code developer environments

GitHub Integration

- **Dependabot** - maintains repository's dependencies automatically
- **CodeQL** - action runs analysis engine to find security vulnerabilities
- **Build & Test** - action runs on each pull request or manually
- **Release** - action runs on new tag, publishing release
- **Docker** - action runs on every commit, publishing Docker images

When it all GOes right

Drivers

CYBERTEC
DATA SCIENCE & POSTGRESQL

NO
ORM
ZONE

Drivers Availability

- **database/sql** is a set of database access interfaces
  - standard de facto creating database applications
  - needs implementation to work
  - read **http://go-database-sql.org** for more information
- **github.com/lib/pq** - pure Go Postgres driver for database/sql
  - is currently in maintenance mode
- **github.com/jackc/pgx** — PostgreSQL driver and toolkit for Go
  - is the default choice nowadays
  - use this even with **database/sql**

pgx vs database/sql

- Use **jackc/pgx** interface (not **database/sql**) when
  - The application only targets PostgreSQL.
  - No other libraries that require **database/sql** are in use.
- Otherwise use **database/sql** with **jackc/pgx/stdlib**
  - compatibility with non-PostgreSQL databases is required
  - when using other libraries that require **database/sql** such as **sqlx** or **gorm**

pgx features
beyond database/sql

- Support for approximately 70 different PostgreSQL types
- Automatic statement preparation and caching
- Batch queries
- Single-round trip query mode
- Full TLS connection control
- Binary format support for custom types
- COPY protocol support for faster bulk data loads

pgx features
beyond database/sql

- Extendable logging support including built-in support for log15adapter, logrus, zap, and zerolog
- Connection pool with after-connect hook
- Listen / notify
- Conversion of PostgreSQL arrays to Go slice mappings
- Hstore support
- JSON and JSONB support
- Large object support

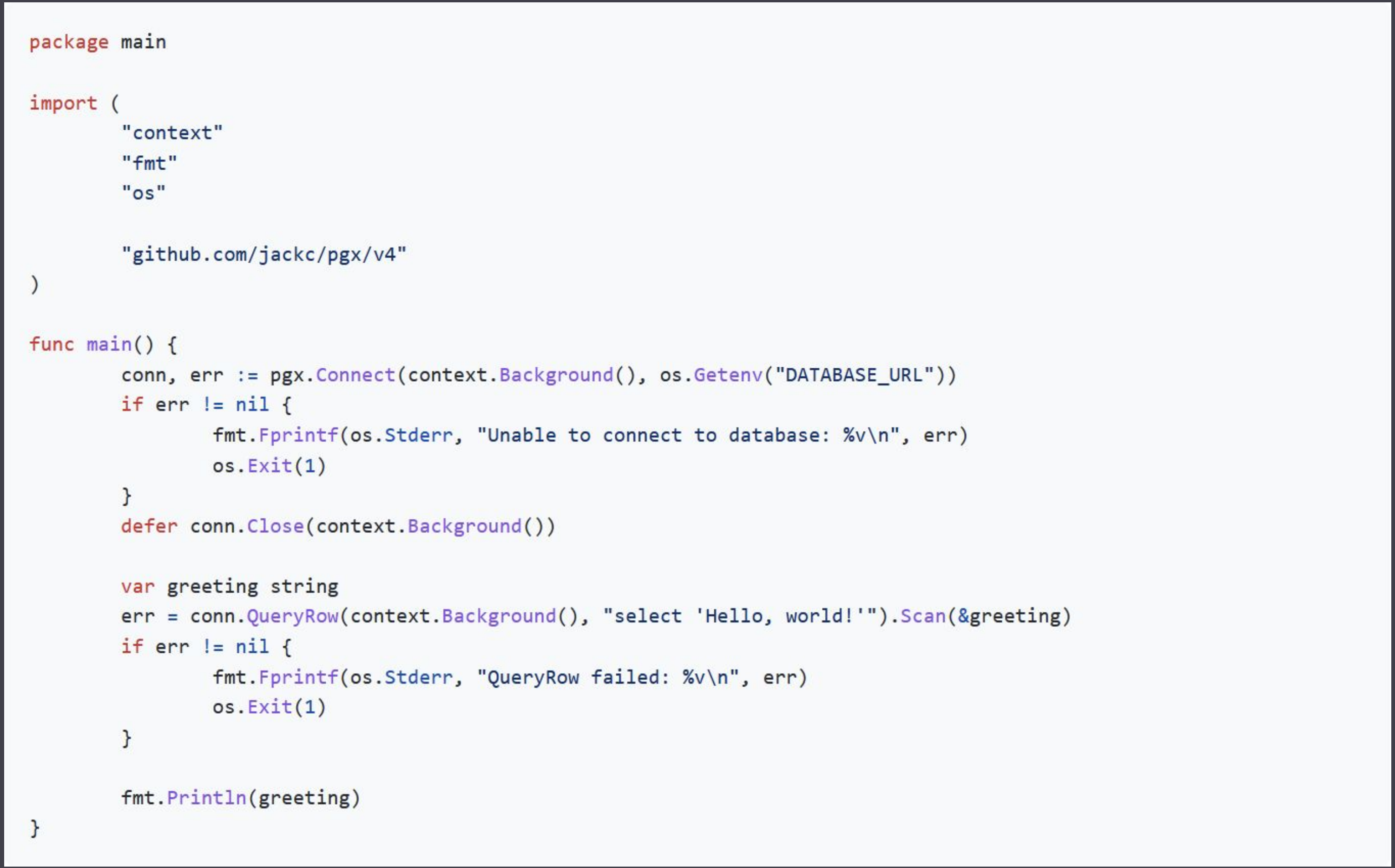
pgx features
beyond database/sql

- NULL mapping to `Null*` struct or pointer to pointer
- Supports database/sql.Scanner and database/sql/driver.Valuer interfaces for custom types
- Notice response handling
- Simulated nested transactions with savepoints

Hello World: database/sql

```go
package main

import (
	"database/sql"
	"fmt"
	"os"

	_ "github.com/jackc/pgx/v4/stdlib"
)

func main() {
	db, err := sql.Open("pgx", os.Getenv("DATABASE_URL"))
	if err != nil {
		fmt.Fprintf(os.Stderr, "Unable to connect to database: %v\n", err)
		os.Exit(1)
	}
	defer db.Close()

	var greeting string
	err = db.QueryRow("select 'Hello, world!'").Scan(&greeting)
	if err != nil {
		fmt.Fprintf(os.Stderr, "QueryRow failed: %v\n", err)
		os.Exit(1)
	}

	fmt.Println(greeting)
}
```

Hello World: jackc/pgx



```go
package main

import (
	"context"
	"fmt"
	"os"

	"github.com/jackc/pgx/v4"
)

func main() {
	conn, err := pgx.Connect(context.Background(), os.Getenv("DATABASE_URL"))
	if err != nil {
		fmt.Fprintf(os.Stderr, "Unable to connect to database: %v\n", err)
		os.Exit(1)
	}
	defer conn.Close(context.Background())

	var greeting string
	err = conn.QueryRow(context.Background(), "select 'Hello, world!'").Scan(&greeting)
	if err != nil {
		fmt.Fprintf(os.Stderr, "QueryRow failed: %v\n", err)
		os.Exit(1)
	}

	fmt.Println(greeting)
}
```

Hello World: pgxpool

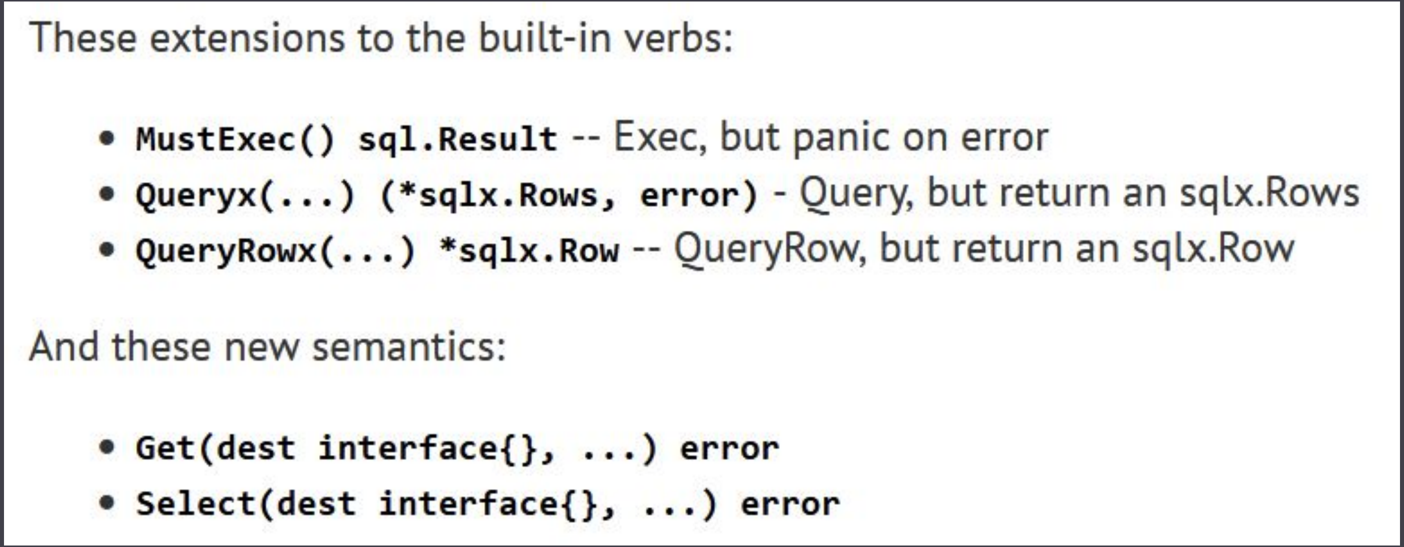
```go
package main

import (
        "context"
        "fmt"
        "os"

        "github.com/jackc/pgx/v4/pgxpool"
)

func main() {
        dbpool, err := pgxpool.Connect(context.Background(), os.Getenv("DATABASE_URL"))
        if err != nil {
                fmt.Fprintf(os.Stderr, "Unable to connect to database: %v\n", err)
                os.Exit(1)
        }
        defer dbpool.Close()

        var greeting string
        err = dbpool.QueryRow(context.Background(), "select 'Hello, world!'").Scan(&greeting)
        if err != nil {
                fmt.Fprintf(os.Stderr, "QueryRow failed: %v\n", err)
                os.Exit(1)
        }

        fmt.Println(greeting)
}
```

When it all GOes right

Useful extensions

CYBERTEC
DATA SCIENCE & POSTGRESQL

Useful Extensions:
database/sql + jmoiron/sqlx



These extensions to the built-in verbs:

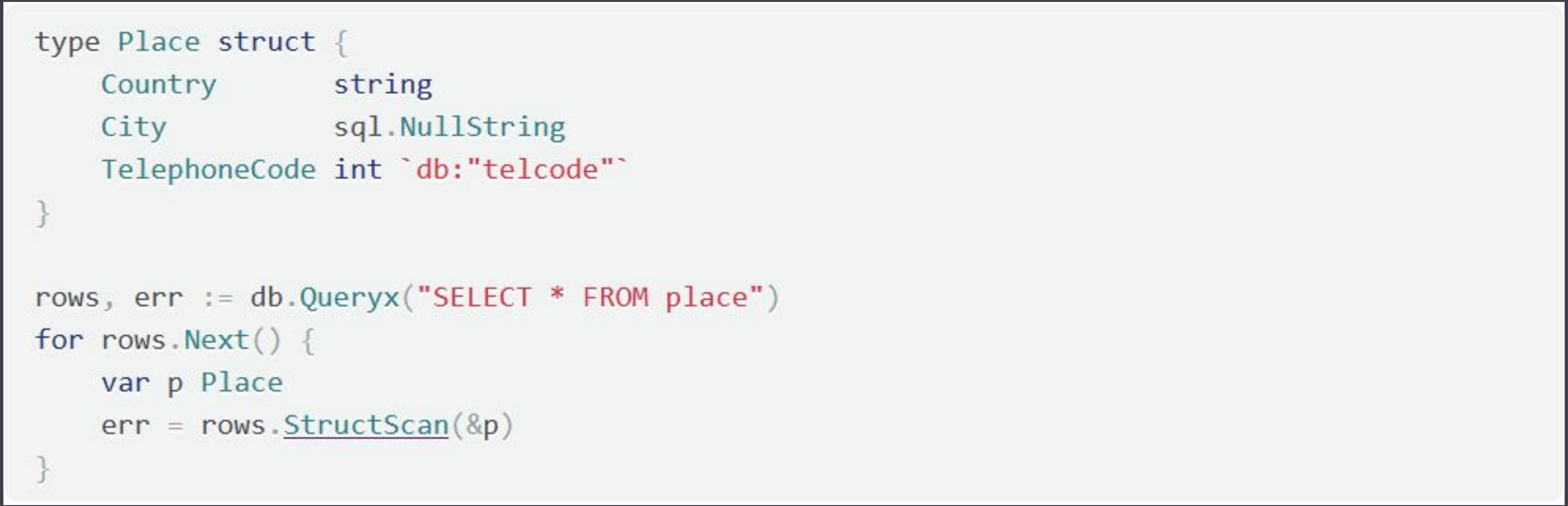
- `MustExec() sql.Result` -- Exec, but panic on error
- `Queryx(...) (*sqlx.Rows, error)` - Query, but return an sqlx.Rows
- `QueryRowx(...) *sqlx.Row` -- QueryRow, but return an sqlx.Row

And these new semantics:

- `Get(dest interface{}, ...) error`
- `Select(dest interface{}, ...) error`

CYBERTEC
The PostgreSQL Database Company

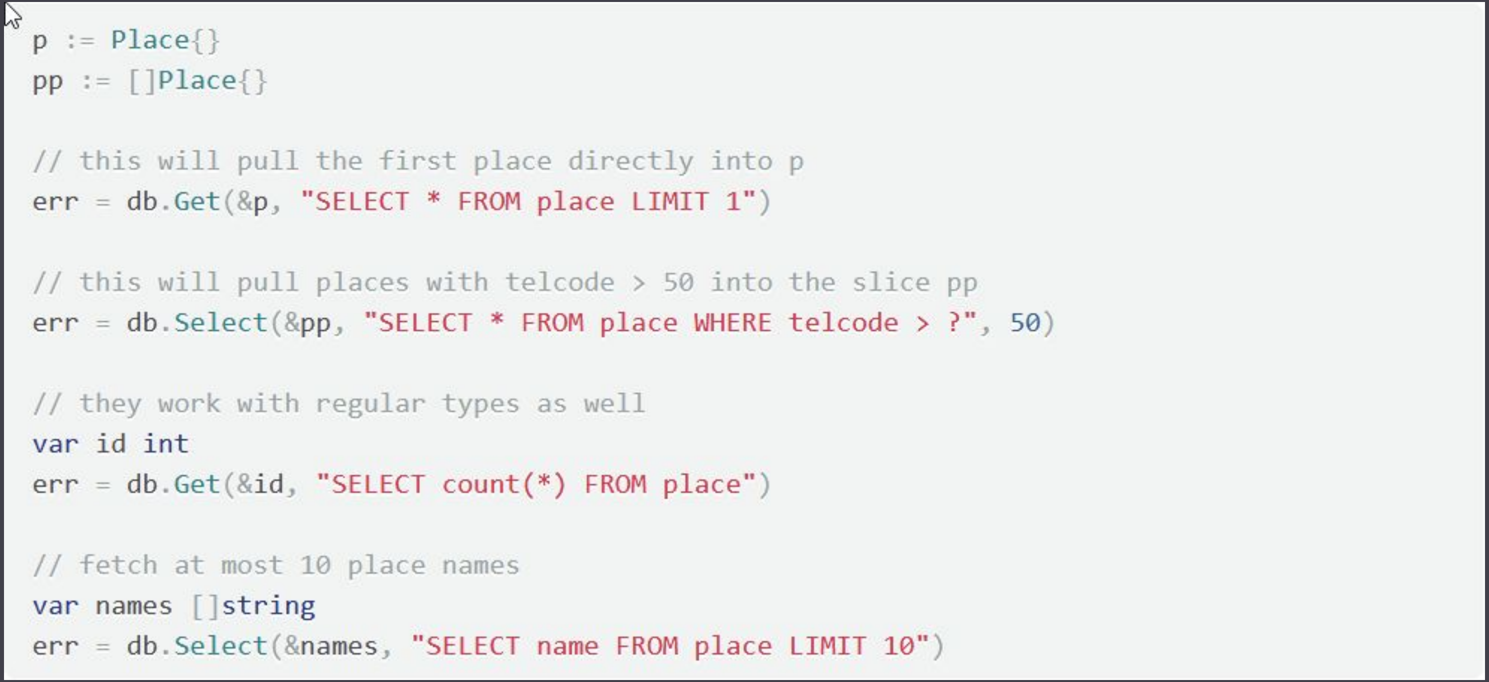
Useful Extensions:
database/sql + jmoiron/sqlx



```
type Place struct {
    Country       string
    City          sql.NullString
    TelephoneCode int `db:"telcode"`
}

rows, err := db.Queryx("SELECT * FROM place")
for rows.Next() {
    var p Place
    err = rows.StructScan(&p)
}
```

CYBERTEC
The PostgreSQL Database Company

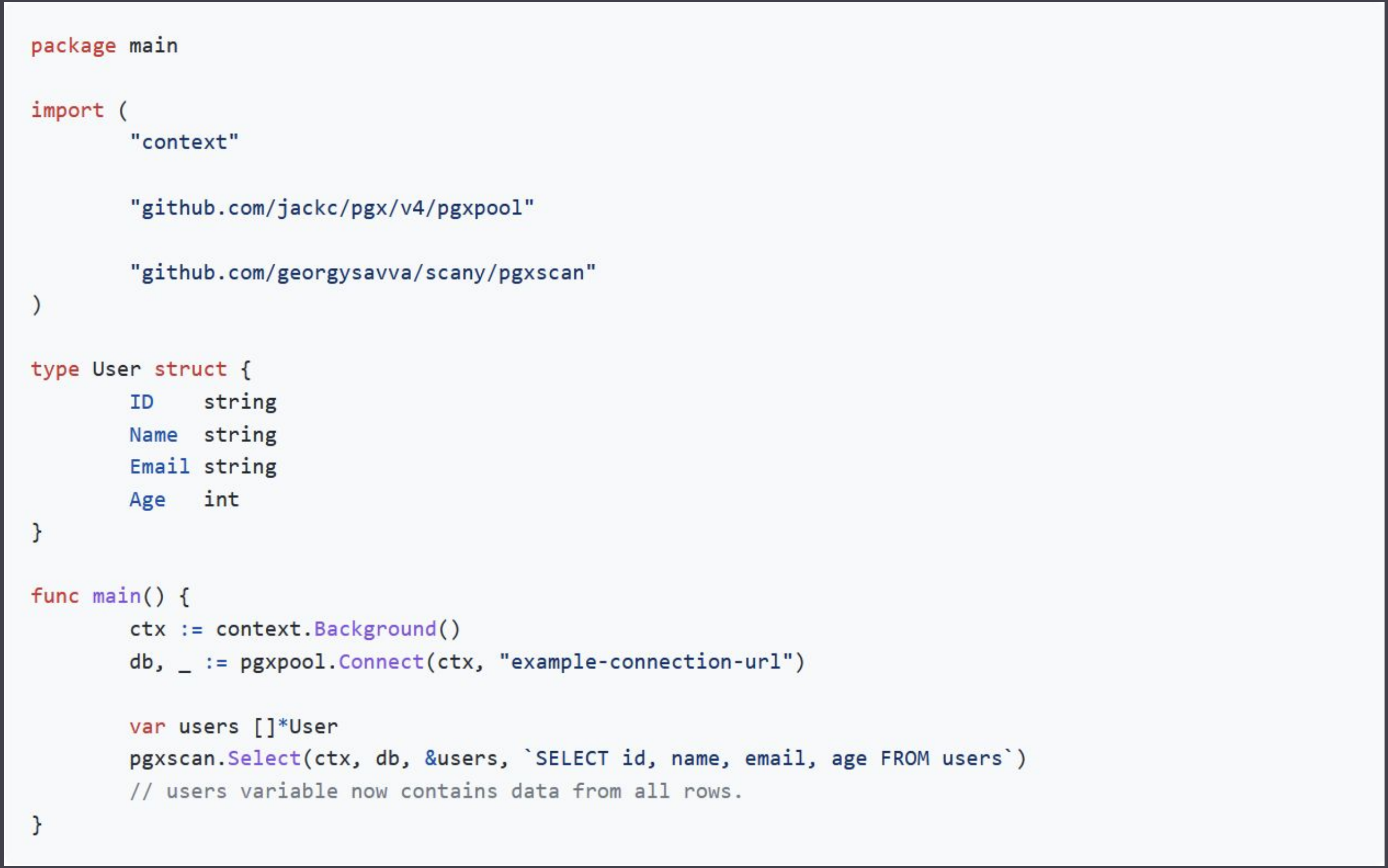
Useful Extensions:
database/sql + jmoiron/sqlx



```go
p := Place{}
pp := []Place{}

// this will pull the first place directly into p
err = db.Get(&p, "SELECT * FROM place LIMIT 1")

// this will pull places with telcode > 50 into the slice pp
err = db.Select(&pp, "SELECT * FROM place WHERE telcode > ?", 50)

// they work with regular types as well
var id int
err = db.Get(&id, "SELECT count(*) FROM place")

// fetch at most 10 place names
var names []string
err = db.Select(&names, "SELECT name FROM place LIMIT 10")
```

CYBERTEC
The PostgreSQL Database Company

Useful Extensions: scany



```go
package main

import (
        "context"

        "github.com/jackc/pgx/v4/pgxpool"

        "github.com/georgysavva/scany/pgxscan"
)

type User struct {
        ID    string
        Name  string
        Email string
        Age   int
}

func main() {
        ctx := context.Background()
        db, _ := pgxpool.Connect(ctx, "example-connection-url")

        var users []*User
        pgxscan.Select(ctx, db, &users, `SELECT id, name, email, age FROM users`)
        // users variable now contains data from all rows.
}
```

When it all GOes right

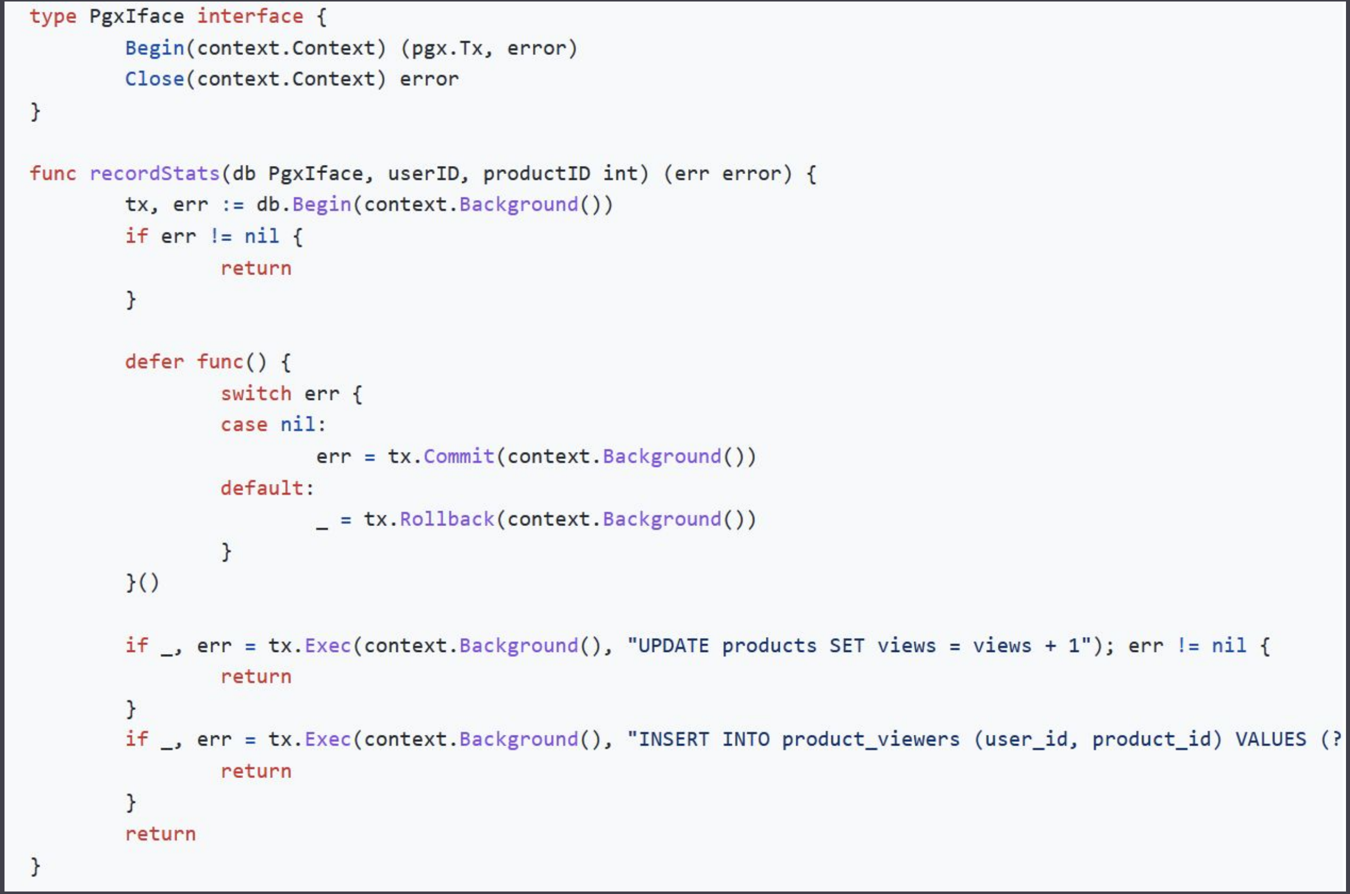
Testing

CYBERTEC
DATA SCIENCE & POSTGRESQL

Testing Approaches

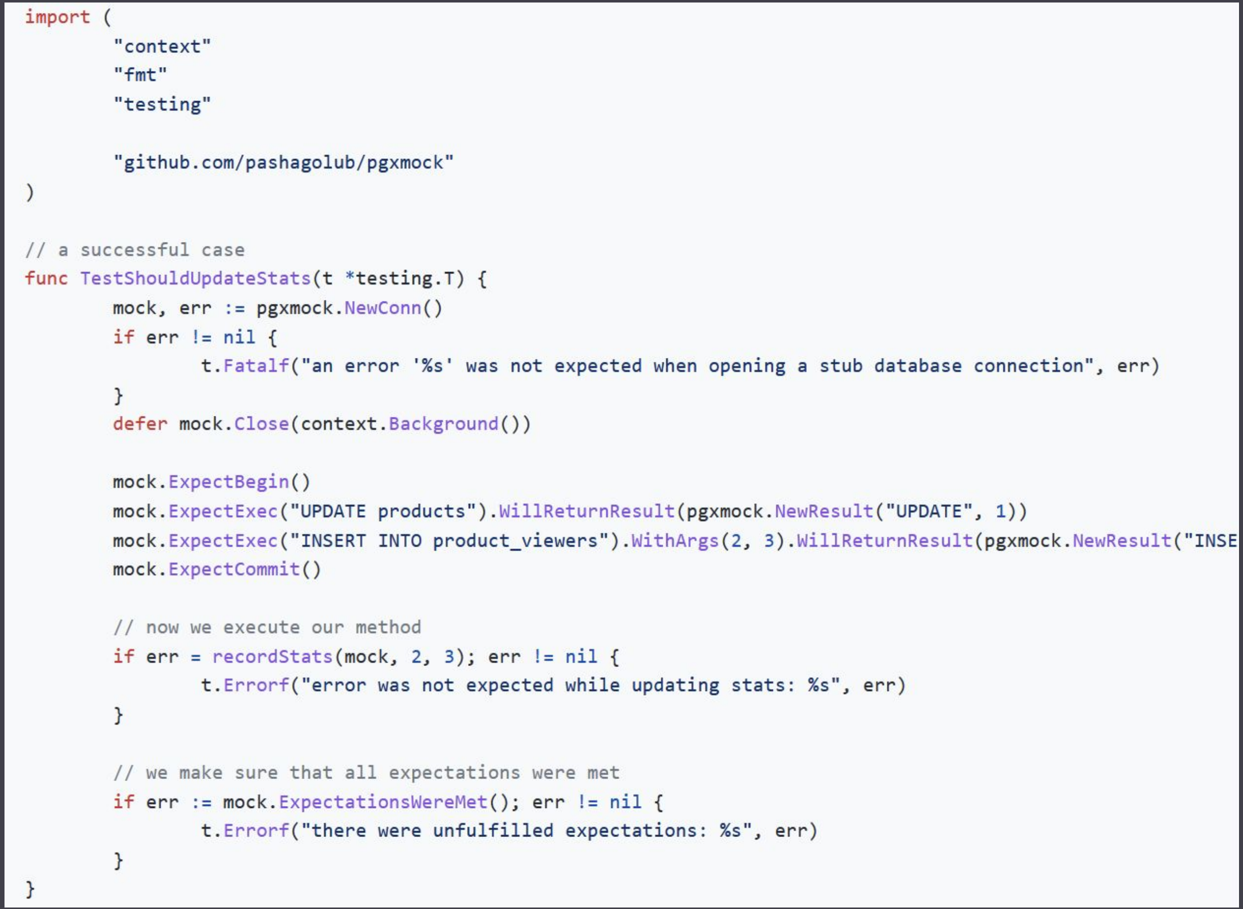
- Real PostgreSQL server
- Mocking libraries
    - **DATA-DOG/go-sqlmock**
    - **pashagolub/pgxmock**
- Mock PostgreSQL wire protocol
    - **jackc/pgmock**
    - **cockroachdb/cockroach-go** Testserver

Mocking Example: pgxmock



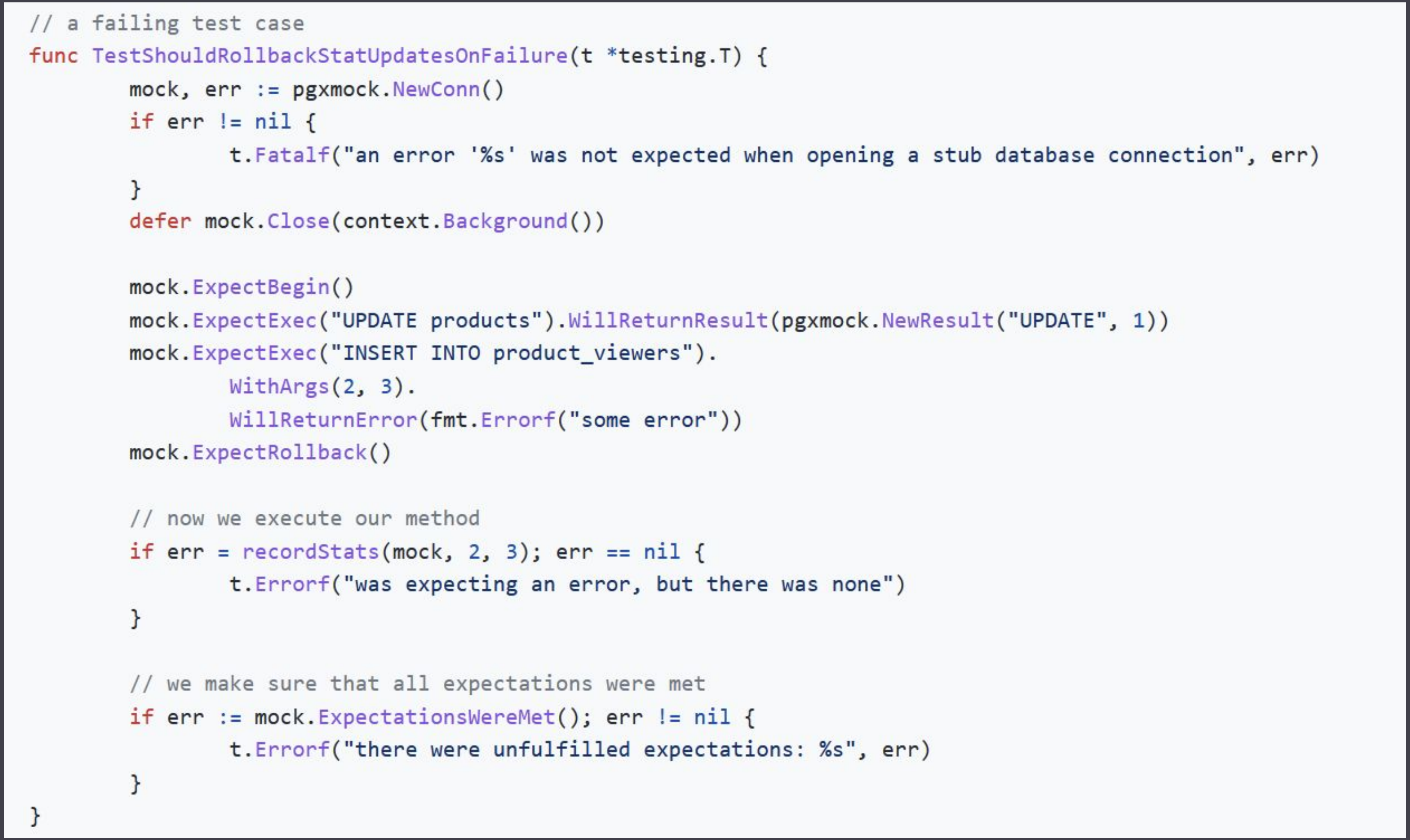
```go
type PgxIface interface {
        Begin(context.Context) (pgx.Tx, error)
        Close(context.Context) error
}

func recordStats(db PgxIface, userID, productID int) (err error) {
        tx, err := db.Begin(context.Background())
        if err != nil {
                return
        }

        defer func() {
                switch err {
                case nil:
                        err = tx.Commit(context.Background())
                default:
                        _ = tx.Rollback(context.Background())
                }
        }()

        if _, err = tx.Exec(context.Background(), "UPDATE products SET views = views + 1"); err != nil {
                return
        }
        if _, err = tx.Exec(context.Background(), "INSERT INTO product_viewers (user_id, product_id) VALUES (?
                return
        }
        return
}
```

Mocking Example: pgxmock



```go
import (
	"context"
	"fmt"
	"testing"

	"github.com/pashagolub/pgxmock"
)

// a successful case
func TestShouldUpdateStats(t *testing.T) {
	mock, err := pgxmock.NewConn()
	if err != nil {
		t.Fatalf("an error '%s' was not expected when opening a stub database connection", err)
	}
	defer mock.Close(context.Background())

	mock.ExpectBegin()
	mock.ExpectExec("UPDATE products").WillReturnResult(pgxmock.NewResult("UPDATE", 1))
	mock.ExpectExec("INSERT INTO product_viewers").WithArgs(2, 3).WillReturnResult(pgxmock.NewResult("INSE
	mock.ExpectCommit()

	// now we execute our method
	if err = recordStats(mock, 2, 3); err != nil {
		t.Errorf("error was not expected while updating stats: %s", err)
	}

	// we make sure that all expectations were met
	if err := mock.ExpectationsWereMet(); err != nil {
		t.Errorf("there were unfulfilled expectations: %s", err)
	}
}
```

Mocking Example: pgxmock



```go
// a failing test case
func TestShouldRollbackStatUpdatesOnFailure(t *testing.T) {
	mock, err := pgxmock.NewConn()
	if err != nil {
		t.Fatalf("an error '%s' was not expected when opening a stub database connection", err)
	}
	defer mock.Close(context.Background())

	mock.ExpectBegin()
	mock.ExpectExec("UPDATE products").WillReturnResult(pgxmock.NewResult("UPDATE", 1))
	mock.ExpectExec("INSERT INTO product_viewers").
		WithArgs(2, 3).
		WillReturnError(fmt.Errorf("some error"))
	mock.ExpectRollback()

	// now we execute our method
	if err = recordStats(mock, 2, 3); err == nil {
		t.Errorf("was expecting an error, but there was none")
	}

	// we make sure that all expectations were met
	if err := mock.ExpectationsWereMet(); err != nil {
		t.Errorf("there were unfulfilled expectations: %s", err)
	}
}
```

When it all GOes right
Takeaways

CYBERTEC
DATA SCIENCE & POSTGRESQL

Takeaways

- Go language is popular, fast, easy and well scaled
- Go is the #1 most in-demand coding language (by Hired)
- 45% of developers use Go to work with databases (go.dev survey)
- You can use your preferred editor or use special IDEs to work
- Kubernetes operators (including Postgres ones) are written in Go
- Go is flexible when working with Postgres: use **sql** or **pgx** interfaces
- Go has powerful programming tools and GitHub/GitLab integration
- Go is backwards compatible. The APIs may grow but not in a way that breaks existing Go 1 code.

Improvement ideas?
User input very much appreciated!

github.com/cybertec-postgresql

github.com/pashagolub

Thanks

Don't be a stranger:

https://www.cybertec-postgresql.com/en/blog/

CYBERTEC
The PostgreSQL Database Company

QUESTIONS

Senior Consultant/Developer

✉ pavlo.golub@cybertec.at

@PavloGolub

CYBERTEC
The PostgreSQL Database Company